Second-Order Masked Lookup Table Compression Scheme

Annapurna Valiveti and Srinivas Vivek

IIIT Bangalore, IN
annapurna@iiitb.org,srinivas.vivek@iiitb.ac.in

Abstract. Masking by lookup table randomisation is a well-known technique used to achieve side-channel attack resistance for software implementations, particularly, against DPA attacks. The randomised table technique for first- and second-order security requires about $m \cdot 2^n$ bits of RAM to store an (n, m)-bit masked S-box lookup table. Table compression helps in reducing the amount of memory required, and this is useful for highly resource-constrained IoT devices. Recently, Vadnala (CT-RSA 2017) proposed a randomised table compression scheme for first- and second-order security in the probing leakage model. This scheme reduces the RAM memory required by about a factor of 2^l, where l is a compression parameter. Vivek (Indocrypt 2017) demonstrated an attack against the second-order scheme of Vadnala. Hence achieving table compression at second and higher orders is an open problem.

In this work, we propose a second-order secure randomised table compression scheme which works for any (n, m)-bit S-box. Our proposal is a variant of Vadnala's scheme that is not only secure but also significantly improves the time-memory trade-off. Specifically, we improve the online execution time by a factor of 2^{n-l}. Our proposed scheme is proved 2-SNI secure in the probing leakage model. We have implemented our method for AES-128 on a 32-bit ARM Cortex processor. We are able to reduce the memory required to store a randomised S-box table for second-order AES-128 implementation to 59 bytes.

Keywords: Masking · S-box · Table compression · Probing leakage model · SNI security · Side-channel attacks · IoT security · Software implementation.

1 Introduction

IoT involves extending connectivity beyond standard devices, such as desktops, laptops, smartphones and tablets, to a range of traditionally non-connectivity-enabled and everyday objects. These devices can communicate and interact over LAN/Internet, and they can be remotely monitored and controlled. Connected devices are part of a framework in which every device talks to other related devices to automate tasks, and to communicate sensed data to users. Technological improvements will make these devices cheaper, smaller and more energy efficient. The embedded devices deployed within IoT are expected to be resource constrained and do not necessarily contain the computing resources necessary to implement strong security protocols. Since IoT devices may be easily physically accessible, it makes them vulnerable to physical implementation-based attacks, such as side-channel attacks [Koc96, KJJ99]. Therefore, there is a need to focus on countermeasures that defend these classes of attacks. In order to protect the implementations of resource constrained IoT devices one must also keep in mind the memory and other storage capacities of these devices.

To counter side-channel attacks against cryptographic implementations, masking is a technique that is popularly used. In masking, a sensitive variable, say $x \in \{0,1\}^n$, is split into $t+1$ shares such that:

$$x = x_1 \oplus \ldots\ldots \oplus x_{t+1}, \tag{1}$$

where any subset of t shares are uniform random and independent of x. Here t is typically referred to as the *masking order*. Though the cost of performing side-channel attacks grows exponentially with the number of shares [CJRR99, PR13, DDF14a], lower masking orders strike a balance between implementation efficiency and security for many real world applications.

Block ciphers consist of both linear transformations and non-linear functions (for e.g., S-boxes). Linear operations are easier to mask since they can be applied on individual shares to produce a shared output. But processing non-linear components of a block cipher implementation in the presence of shares is relatively expensive. The overhead compared to an unmasked implementation is $O(t)$ for linear operations, while it is $O(t^2)$ for non-linear operations [ISW03].

There are many approaches suggested in the literature to securely mask S-box implementations at arbitrary orders. These S-box masking schemes may be categorised as being circuit/computation-based [ISW03] or table-based schemes [Cor14]. The polynomial-based and bitsliced masked implementations belong to the former category [CGP$^+$12, CRV15, CPRR15, GRVV17, JS17, GR17]. The polynomial-based and bitsliced schemes are suitable for masking at higher orders due to better timing and RAM memory requirements than lookup table-based schemes [Cor14, CRZ18, GTP$^+$20]. However, lookup table-based methods are more suitable for low orders due to better efficiency and as they also typically allow *pre-processing* to achieve better *online* execution time. By "pre-processing" we mean the possibility of choosing all but one shares of the secret variables a priori and performing any associated computation offline, before actual execution of the scheme. The remaining processing time using secret variables is referred to as the "online" time.

Lookup Table-based S-box Masking Schemes. The first randomised lookup table-based scheme with a formal analysis and secure at first-order was proposed by Chari *et al.* in [CJRR99]. In this method, to implement an (n, m) S-box S, a randomised table T is created in RAM using a share of the S-box input as the input mask, and then table entries are also protected with an output mask. The masked table T is computed in RAM memory as follows:

$$T(u) = \mathrm{S}(u \oplus x_1) \oplus y_1, \quad 0 \leq u \leq 2^n - 1, \tag{2}$$

where the input share x_1 is uniform random and independently chosen, while the input share x_2 is computed as $x = x_1 \oplus x_2 \in \{0,1\}^n$, y_1 is an output mask $\in \{0,1\}^m$ which is chosen uniform random and independent of x. The second output share y_2 is computed from T as $y_2 = T(x_2)$. The above defined S-box masking scheme is secure against first-order attacks as no intermediate variable is statistically dependent on the secret x, but the pair x_1 and x_2 together leak the secret x.

The amount of RAM memory required to store the randomised table T according to (2) is $m \cdot 2^n$ bits. The second-order lookup table-based schemes were proposed by Schramm and Paar [SP06], and Rivain, Dottax and Prouff [RDP08]. These schemes require a temporary table in RAM of size $2 \cdot m \cdot 2^n$ and $m \cdot 2^n$ bits, respectively, while the higher-order lookup table-based scheme suggested by Coron [Cor14] requires $(2t+1) \cdot m \cdot 2^n$ bits for t-th order security. Later, Coron *et* al. in [CRZ18] proved that the scheme from [Cor14] is indeed t-SNI secure with number of shares $n = t+1$ instead of $n = 2t+1$ in the original scheme. Hence an improvement by a factor of two was achieved for both memory and randomness complexity and the runtime was improved by a factor of 4.8 for AES [FIP] with $t = 6$. Therefore, the higher-order masked lookup table scheme [CRZ18] requires $(t+1) \cdot m \cdot 2^n$ bits for t-th order security. Recently, Guo *et al.* in [GTP$^+$20] proposed a higher-order masking scheme that is also claimed to be resistant against *horizontal attacks*. Note that the previously mentioned schemes are not known to be secure against horizontal attacks. This scheme manages to achieve memory compression by approximately a factor of two compared to [CRZ18]. But this scheme cannot support pre-processing since the lookup table compression is dependent on all input shares. Essentially, all the computations happen during the online phase.

Moreover, we need to note that the above calculations are for masking only a single S-box. In a typical block cipher that uses S-boxes, there will be several S-box lookups. For instance, for AES-128 [FIP], in each round there are 16 S-box lookups, and there are 10 rounds. So in total, there will be 160 S-box lookups. If the full pre-processing ability of the table-based masking schemes is needed to be used, then for first- and second-order security, we need to store all the 160 masked tables at once in RAM for a total cost of 40 KB. This could be expensive for tiny IoT devices where there are other applications contending for the RAM memory. Hence there is a motivation to achieve trade-off for the online execution time vs. the RAM memory for pre-processing even at small orders, while ensuring that the online execution time is better than the circuit-based masking schemes.

Masked Table Compression Schemes. There has been an increasing demand to design countermeasures against side-channel attacks for resource constrained devices. Rao *et al.* [RRST02] recommended a table compression scheme that requires $m \cdot 2^{n-1}$ bits to store the masked table in RAM at first-order security. For the case of AES this means that the masked table for an S-box needs only 128 bytes. This method can also be extended to compress the table size to $\approx m \cdot 2^n/l$ bits for an (n, m)-bit S-box, where the *compression parameter* l is such that $1 \leq l \leq n-1$. Recently, Vadnala [Vad17] proposed a table compression schemes for first-order and second-order that significantly improved upon the method of [RRST02]. The memory requirement for the first-order case was reduced to $\approx m \cdot 2^{n-l} + (n-l) \cdot 2^l$ bits, where $1 \leq l \leq (n-1)$. For the second-order case, it was shown to be $\approx m \cdot 2^{n-l} + (n-l+1) \cdot 2^l$ bits. Reasonably efficient first- and second-order masked implementations of AES-128 were obtained using only about 40 bytes of RAM memory per S-box [Vad17]. The proposed schemes were argued to be secure in the probing leakage model [ISW03].

Later, Vivek [Viv17] demonstrated an attack against the second-order table compression scheme of [Vad17]. Section 3 describes the second-order scheme of [Vad17] and discusses the attack of [Viv17]. Hence designing masked table compression schemes at second and higher orders has remained an open problem.

Our Contribution. In this work we propose a secure second-order masked S-box lookup table compression scheme for arbitrary S-boxes. Our proposal is a variant of the second-order scheme of [Vad17] that is not only secure but significantly improves upon the online execution time by a factor 2^{n-l}. Concretely, we also use the technique of [Vad17] to pack 2^l entries of the uncompressed table into a single entry of the first compressed table. Then, securely using $n-l$ most significant bits of x, we generate another masked table of size 2^l that is then securely accessed using the remaining l bits. The second-order scheme of [Vad17] does not allow pre-processing since all the shares of the secret variable are needed for most of the computation steps. This limitation is inherited from the second-order S-box masking scheme of Rivain, Dottax and Prouff [RDP08] on which the second-order scheme of [Vad17] is based.

Our first contribution is to propose a variant of the second-order masking scheme of [RDP08] that allows pre-processing, where the final share is needed only in the last step (see Section 2). We prove that our variant is second-order secure in the probing leakage model under compositions, *a.k.a.* 2-SNI probing secure [BBD$^+$16]. The original scheme of [RDP08] was proved to be second-order probing secure only for balanced S-boxes (see

Definition 1). But, this is not a restriction since the balancedness property holds for all the known cryptographic S-boxes. However, for the sake of completeness, we extend the scheme of [RDP08] to *arbitrary* S-boxes that are not necessarily balanced. The cost incurred for this is only logarithmic (in the size of the table) amount of randomness and memory for storing this randomness. We achieve this by generating (a part of) the randomness *on-the-fly* using a pairwise-independent PRG even when it needs to be reused (see Section 2.2). We would like to note that in spite of the slight increase in the randomness and memory overhead compared to the original RDP scheme, the randomness and memory complexity is still nearly 3 times better than the table-based masking scheme of Coron [Cor14] instantiated at second-order security. Then, we extend this variant to the table compression scheme in Section 4 and we prove our constructions to be 2-SNI secure in the probing leakage model.

As far as the technique to make our table compression scheme second-order secure is concerned, the immediate observation is to protect against the attack mentioned in [Viv17]. This attack is possible since the same mask was used in [Vad17] for different rows of each table. To overcome the attack, one would naturally think of only masking each entry of the second intermediate table with a different mask. But, as we have observed, we even need to randomise the rows of the offline table as well. On top of it, we need to even secure the index of the final output share (see Algorithm 6). The RAM memory required for the proposed compression scheme to mask an (n, m) S-box is $\approx m \cdot (n-l+1) + 2^l \cdot (n-l+m) + m \cdot (2^{n-l} + 2^l)$ bits. We would like to note that, to the best of our knowledge, there is no direct approach for formal verification of table-based schemes using the existing tools such as [Cor18, BGR18]. These tools natively support circuit-based schemes. However, in [BBD$^+$15], the authors managed to verify the table-based scheme [SP06] indirectly. It will be an interesting future work to extend their technique to formally verify the security of our proposed scheme.

Finally, in Section 5, we report the performance of our proposed table compression scheme on a 32-bit ARM micro-controller. We reduce the RAM memory required per S-box to mask AES-128 at second-order security to 59 bytes (see Table 3 on Page 18) and that of second-order masked PRESENT 80-bit key variant to 9 bytes (see Table 4 on Page 18). We compare the online execution time of our second-order compression of AES-128 with that of the second-order masked implementations of Rivain and Prouf [RP10] and bitsliced AES, where the latter schemes do not take any advantage of pre-processing. As Tables 5 and 7 suggest, we achieve improved online execution time over both [RP10] and bitsliced implementations for the compression parameter values up to $l = 3$. For completeness, we compare the implementation results of second-order compression of PRESENT with that of second-order circuit-based implementation of PRESENT using [CRV15]. As Tables 6 and 9 indicate, the online execution time is better than [CRV15] for the compression parameter values $l = 1, 2$.

2 Variant of 2-O Scheme of [RDP08] with Pre-processing

In this section, we present our variant of the second-order secure randomised table-based S-box masking scheme of Rivain, Dottax and Prouff [RDP08, Section 3.1, Algorithm 2]. Unlike the former scheme, our variant allows pre-computing of the masked table offline as the third share is needed only in the last step.

The second-order scheme in [RDP08] makes use of all the three input shares, say, x_1, x_2 and x_3, as inputs to the randomised table computation. Because of this requirement the randomised table, say, T, has to be evaluated only during the online execution phase and hence cannot benefit from any offline pre-processing. We modify the table construction in such a way that it can be computed independent of the secret, i.e., using only two shares x_1 and x_2. Interestingly, the improvement is possible with only a small tweak to the original algorithm. Hence, instead of recalling the scheme from [RDP08], we right away present our variant in Algorithm 1. The change from the original scheme is that the variables υ and x_3 in Algorithm 1 have been swapped. We have tried to be consistent with the notation in [Viv17]. The mapping between the notations of [RDP08] and Algorithm 1 is summarised in Remark 1.

Remark 1. The variables r_1, r_2, $\tilde{x}$, r_3, r', s_1, s_2 in [RDP08, Algorithm 2] correspond to x_1, x_2, x_3, υ, d, y_1, y_2, respectively, in Algorithm 1.

Algorithm 1: Variant of [RDP08, Algorithm 2]: second-order secure masked S-box computation with *pre-processing*.

```
Input  :
    • Input shares $x_1$, $x_2$, $x_3$ such that $x = x_1 \oplus x_2 \oplus x_3$.
    • An $(n, m)$ S-box lookup table S.
Output : Three output shares: $y_1$, $y_2$, $y_3$, such that $\mathrm{S}(x) = y_1 \oplus y_2 \oplus y_3$.
1 $\upsilon \xleftarrow{\$} \{0,1\}^n$
2 $y_1 \xleftarrow{\$} \{0,1\}^m$
3 $y_2 \xleftarrow{\$} \{0,1\}^m$
4 $d \longleftarrow (x_1 \oplus \upsilon) \oplus x_2$
5 for $a \leftarrow 0$ to $2^n - 1$ do
6     $b \longleftarrow a \oplus d$
7     $T(b) \longleftarrow (\mathrm{S}(\upsilon \oplus a) \oplus y_1) \oplus y_2$
8 end
9 $y_3 = T(x_3)$
```

Note that in order to construct the table T, only two out of the three input shares of x are needed. Since two out of three shares can be chosen uniformly at random, the construction of T is independent of x. The third input share x_3 is used only while accessing the table in the last step. Like the original scheme, a random variable υ is used to mask x_1 while combining with x_2. Further, the same υ is used to shift the table and then table entries are protected using the output masks y_1 and y_2. Finally, we get the final share when the table is accessed at $T(x_3) = \mathrm{S}(x_1 \oplus x_2 \oplus x_3) \oplus y_1 \oplus y_2 = \mathrm{S}(x) \oplus y_1 \oplus y_2$.

2.1 2-SNI Security Proof

We next prove that our scheme is second-order secure in the probing leakage model in the presence of compositions, i.e., 2-SNI secure (see Definition 2) only if the S-box is balanced. The original proposal was only known to be second-order secure in the probing leakage model without compositional security guarantees. In the rest of the paper, we use only the 2-SNI security notion. To define the balancedness property formally,

Definition 1. Balanced S-box. An (n, m) S-box S, $m \leq n$, is balanced if every output word is the image under S of exactly 2^{n-m} input words.

Let us also recall the compositional security notion of Strong Non-Interference (SNI) from [BBD$^+$16]. A gadget is said to be t-SNI secure if any set of t intermediate variables and/or output shares are probed, then probed variables can be simulated only with the number of shares (of each input variable) equal to the number of probes on the intermediate variables (including the input shares). Formally,

Definition 2. Strong Non-Interference (t-SNI) Security [BBD+16]. Let x be an input split into n shares. Let G be a gadget which takes shares of the input x, i.e., x_i, and outputs the shares y_i. Let G be probed with $t < n$ probes, out of which, say, a set of t_1 intermediate variables (including input shares) and a set of t_2 output shares are probed, and $t = t_1 + t_2$. Then G is said to be t-SNI secure if the set of t probes can be perfectly simulated by using only t_1 shares of the input x.

For completeness, we have tabulated the intermediate variables (including inputs, intermediates and outputs) of Algorithm 1 in Table 1. Further, we partition the variables into two sets: Set_1 consists of input shares and intermediate variables, and Set_2 having only output shares. Each I_k represents input or output or intermediate variables as defined in Table 1. The basic idea behind the 2-SNI security proof of our variant is as follows: since two output shares (y_1 and y_2) to mask $\mathrm{S}(x)$ are chosen at random, any two out of three output shares are easy to simulate without using any input share. As Algorithm 1 uses pre-processing phase that does not involve x_3, it can be observed that any pair of variables from the pre-processing phase can be simulated using a maximum of two input shares x_1 and x_2, independent of the secret x. Even though the same output masks are used across the table, as explained below in the formal proof, the balanced S-box property comes handy to show the simulation. Pairs of variables depending on all three input shares can be simulated using the unprobed randomness present in the variables.

Table 1: List of variables of Algorithm 1.

j	I_j
Input and intermediate variables	
1	input shares x_1, x_2 and $x_3 = x \oplus x_1 \oplus x_2$
2	a, a constant
3	v
4	$v \oplus a$
5	$x_1 \oplus v$
6	$d = (x_1 \oplus v) \oplus x_2$
7	$a \oplus d = a \oplus ((x_1 \oplus v) \oplus x_2)$
8	$\mathrm{S}(v \oplus a)$
9	$\mathrm{S}(v \oplus a) \oplus y_1$
10	$T(b) = (\mathrm{S}(v \oplus a) \oplus y_1) \oplus y_2$
Output variables	
11	y_1
12	y_2
13	$y_3 = T(x_3) = \mathrm{S}(x) \oplus y_1 \oplus y_2$

Theorem 1. *Algorithm 1 is 2-SNI secure if the S-box is balanced.*

Proof. Formally, in order to prove the scheme to be 2-SNI secure, note that the gadget is the S-box S, which takes an input x in the form of three shares $x_1, x_2, x_3 = x \oplus x_1 \oplus x_2$, and outputs y_1, y_2, $y_3 = \mathrm{S}(x) \oplus y_1 \oplus y_2$. For convenient representation, we have partitioned Table 1 into two sets :

$$\text{Set}_1 := \text{input and intermediate variables,}$$
$$\text{Set}_2 := \text{output shares.}$$

The following are the possible types of combinations for probing pairs of intermediate variables (including input shares, intermediate variables and output shares).

1. *Pair of output shares*: if the probed output shares are (y_1, y_2), then this pair can be assigned random values as this would have happened in the actual implementation. If the probed pair of output shares are either (y_2, y_3) or (y_1, y_3), then the pair can be simulated with the help of the fact that y_1 or y_2, respectively, is not probed. Hence, any pair of output shares can be simulated without the knowledge of input shares.

2. *One input/intermediate and one output share*: similar to Step 1, the probed output share can be assigned a uniformly chosen random value.

    - Probed intermediate variable depending on at most one input share (including constants, sampled randomness and input shares) is trivial to simulate. It can be observed from the Table 1 that the intermediate variable relying on two input shares is always associated with the uniform random and independent value v. Since v can not be probed in this setting, the intermediate variable having two input shares can be assigned a uniform random and independent value without the knowledge of any input share.
    - Since the pair of output masks y_1 and y_2 are common for the final output share and any row of the table T, to simulate the pair ($y_3 = \mathrm{S}(x) \oplus y_1 \oplus y_2$ and $\mathrm{S}(v \oplus a) \oplus y_1) \oplus y_2$), y_3 can be assigned a uniform random and independent value. Then,

    $$y_1 \oplus y_2 = y_3 \oplus \mathrm{S}(x),$$
    $$\mathrm{S}(v \oplus a) \oplus y_1 \oplus y_2 = \mathrm{S}(v \oplus a) \oplus y_3 \oplus \mathrm{S}(x).$$

    Since the S-box is balanced, the variable $\mathrm{S}(v \oplus a)$ is a uniform random and independent of y_3 and x since the input to the S-box, $v \oplus a$, satisfies the same property (see Definition 1). Therefore, the probed pair can be simulated without the knowledge of any input share. We can conclude that any pair of intermediate variables in these cases can be simulated with a maximum of one input share.

3. *Two intermediate variables*: for this case, we need to prove that $\mathrm{Set}_1 \times \mathrm{Set}_1$ is independent of x. Since pre-processing is independent of x, all the intermediate variables except the last table access in Step 9 can be simulated using at most two input shares. To assign values to the pair having the final input share x_3 and any other intermediate variable from Set_1, as discussed in Step 2, the probed pair can be assigned values using the unprobed randomness v, and x_3. Hence, we can conclude that the simulation in this setting can be done independent of x.

This proves that Algorithm 1 is 2-SNI secure. □

2.2 Variant of [RDP08] for Arbitrary S-box

It may be observed that using common output masks across the rows of the masked table demands the S-box to be balanced in the proof of Algorithm 1. In order to extend our variant scheme to any arbitrary S-box, the idea is to randomise the table with different output mask for each row. But to use distinct masks per row, the required randomness will be *exponential* in n. Moreover, the generated randomness has to be stored in order to retrieve one of them as the final output share. It may be noticed that if each row uses distinct output mask y_1 and output masks are never combined together in the scheme, then it is sufficient to generate y_1's that are *pair-wise independent* for a second-order secure implementation.

Therefore, to reduce the randomness complexity and also to reduce the memory complexity of storing this randomness, we use a slightly tweaked variant of the *subset-sum* technique of generating pair-wise independent randomness as suggested in [Viv17, Section 2]. This variant is same as the one suggested in [TV09] that generates *3-wise independent* random values. This construction to compute the output masks y_1's helps to bring down the randomness complexity of T from 2^n to $n+1$ which is linear in n. Further, to reduce the memory complexity, we do not store the output masks. Instead, we generate the output masks *on-the-fly* as part of the construction and the final share y_1 is also re-constructed on-the-fly. An interesting feature of the subset-sum technique which comes handy in our SNI proofs is that every intermediate variable of subset-sum is an output. Hence there are no intermediate variables that are not outputs and that can be probed. So we do not need the robustness property for the PRG [IKL+13, CGZ19]. We recall the 3-wise independent subset-sum technique in Algorithm 2. By $bits_k(i)[j]$ we mean the j^{th} bit from the least significant position in a k-bit binary representation of i.

Algorithm 2: s-sum: 3-wise independent PRG [TV09].

```
Input  :
    • $k$, and $i \in \{0,1\}^k$. // compute subset-sum for k-bit input index $i$
    • $\gamma_0, \ldots, \gamma_k \in \{0,1\}^m$. // input seed for PRG
Output : $y \in \{0,1\}^m$
1 $y = \gamma_k$
2 for $j \leftarrow 0$ to $k-1$ do
3     if $bits_k(i)[j] \neq 0$ then
4         $y = y \oplus \gamma_j$
5     end
6 end
7 return $y$
```

Using Algorithm 2, Algorithm 3 presents the second-order table based scheme that works for any arbitrary S-box. The 2-SNI proof argument for Algorithm 3 is very similar to the one provided for Algorithm 1 with the only difference that any row of the Table T along with y_3 can be simulated *without* the *S-box balancedness* property (see Definition 1). To be precise, probing the output share y_3 along with any other table entry $T(i)$ can be simulated without using input shares as the pair-wise independent output mask helps to randomise the table row. As already mentioned above, pair-wise independence of output masks is sufficient for 2-SNI security since first output masks are not combined as part of Algorithm 3. This idea of using pair-wise independent output masks presented in Algorithm 3 serves as the basis to propose the second-order masked table compression scheme that works for any arbitrary S-box in Section 4.

3 Recap of 2-O Compression Scheme from [Vad17]

In this section, we recall the second-order masked table compression scheme from [Vad17, Section 3] . We follow the notation used in [Viv17, Remark 1, Section 2].

The table compression method will choose a compression parameter l such that $1 \leq l \leq n-1$. The extreme values $l=0$ and $l=n$ may also be allowed with the natural interpretation of vacuous objects. Define the functions $\mathrm{S}_i : \{0,1\}^{n-l} \rightarrow \{0,1\}^m$, for $0 \leq i \leq 2^l - 1$, as

$$\mathrm{S}_i(u) = \mathrm{S}(u \parallel i), \quad \forall u \in \{0,1\}^{n-l}. \tag{3}$$

Algorithm 3: Second-order secure masked S-box with *pre-processing* for arbitrary S-box.

Input :

- Input shares x_1, x_2, x_3 such that $x = x_1 \oplus x_2 \oplus x_3$.
- An (n, m) S-box lookup table S.

Output : Three output shares: y_1, y_2, y_3, such that $\mathrm{S}(x) = y_1 \oplus y_2 \oplus y_3$.

```
1  $v \xleftarrow{\$} \{0,1\}^n$
2  for $i \leftarrow 0$ to $n$ do
3      $\gamma_i \xleftarrow{\$} \{0,1\}^m$
4  end
5  $y_2 \xleftarrow{\$} \{0,1\}^m$
6  $d \longleftarrow (x_1 \oplus v) \oplus x_2$
7  for $a \leftarrow 0$ to $2^n - 1$ do
8      $b \longleftarrow a \oplus d$
9      $z \longleftarrow \text{s-sum}(n, b, \gamma_0, \ldots, \gamma_n)$
10     $T(b) \longleftarrow (\mathrm{S}(v \oplus a) \oplus z) \oplus y_2$
11 end
12 $y_1 = \text{s-sum}(n, x_3, \gamma_0, \ldots, \gamma_n)$
13 $y_2 = y_2$
14 $y_3 = T(x_3)$
```

In order to reduce the space required to store the masked table T in [RDP08], the idea is to pack 2^l table values into a single entry of a table T_1. Packing is secured by picking uniformly distributed random values r_i's independent of x:

$$r_i \xleftarrow{\$} \{0,1\}^{n-l}, \quad 0 \le i \le 2^l - 1. \tag{4}$$

Let *var* be an n-bit variable such that:

$$var := var^{(1)} \parallel var^{(2)}, \tag{5}$$

where $var^{(1)} \in \{0,1\}^{n-l}$ and $var^{(2)} \in \{0,1\}^{l}$. Each entry of T_1 is defined as:

$$T_1(b^{(1)}) := \left(\left(\bigoplus_{0 \le i \le 2^l - 1} \mathrm{S}_i(x_3^{(1)} \oplus a^{(1)} \oplus r_i) \right) \oplus y_1 \right) \oplus y_2, \tag{6}$$

where

$$b^{(1)} = a^{(1)} \oplus ((x_1^{(1)} \oplus v^{(1)}) \oplus x_2^{(1)}), \tag{7}$$

for all $0 \le a^{(1)} \le 2^{n-l} - 1$, $v^{(1)} \in \{0,1\}^{n-l}$ and r_i as in (4).

After computing T_1 as described in (6), the final share y_3 can be obtained as:

$$y_3 = T_1(v^{(1)} \oplus r_{x^{(2)}}) \oplus \bigoplus_{0 \le j \le 2^l - 1,\ j \ne x^{(2)}} \mathrm{S}_j(x^{(1)} \oplus r_{x^{(2)}} \oplus r_j).$$

But y_3 cannot be computed directly as indicated above as the secret $x^{(2)}$ is directly manipulated. Therefore, we construct another Table T_2 by accessing T_1 with the random value v and the same r_i's used in T_1 construction. Then, securely combine the value in (6) with S-box lookup functions S_i to obtain the enries of T_2. Shares of $x^{(2)}$ are used to index

T_2 entries and they are combined using an independent random value $\upsilon^{(2)}$ (see (5)). Each entry of Table T_2 is defined as follows:

$$T_2(b^{(2)}) := (T_1(\upsilon^{(1)} \oplus r_{(x_3^{(2)} \oplus a^{(2)})})) \oplus \bigoplus_{0 \le j \le 2^l-1,\, j \ne a^{(2)}} \mathrm{S}_{(x_3^{(2)} \oplus j)}((((x_3^{(1)} \oplus r_{(x_3^{(2)} \oplus a^{(2)})}) \oplus x_1^{(1)}) \oplus r_{(x_3^{(2)} \oplus j)}) \oplus x_2^{(1)}),$$

where,

$$b^{(2)} := a^{(2)} \oplus ((x_1^{(2)} \oplus \upsilon^{(2)}) \oplus x_2^{(2)}) \quad \forall\, 0 \le a^{(2)} \le 2^l - 1. \tag{8}$$

Finally, Table T_2 is accessed at $\upsilon^{(2)}$ to obtain $y_3 = \mathrm{S}(x) \oplus y_1 \oplus y_2$.

Instead of storing T_2, the output shares can be computed *on-the-fly* as explained in [Vad17, Section 3]. Using two registers, say R_0 and R_1, the output share y_3 is stored in R_k, where k is an independent and random bit. The loop iterates through all possible 2^l values and saves the result in one of the two registers based on the output of the first-order secure comparison among the shares [RDP08, Appendix]. Finally, R_k contains the third output share y_3. Note that in this scheme, as in the original RDP scheme, all the shares must be present from the beginning. Hence there is no scope for pre-processing.

3.1 Second-Order Attack from [Viv17]

Vivek [Viv17] presented attacks on both variants of the second-order table compression scheme of [Vad17]. These attacks target the dependency that many pairs of intermediate variables depend on the secret which is a violation of the second-order security.

Note that T_2 consists of 2^l entries. Let i and j represent two indices into table T_2. The following holds for all i and $j \in \{0,1\}^l$, $i \ne j$:

$$T_2(i) \oplus T_2(j) = \mathrm{S}_{(i \oplus x^{(2)} \oplus \upsilon^{(2)})}(x^{(1)}) \oplus\ \mathrm{S}_{(j \oplus x^{(2)} \oplus \upsilon^{(2)})}(x^{(1)}). \tag{9}$$

The above relation shows that any pair of values from T_2 jointly depends on $x^{(1)}$, the higher order $n-l$ bits of the secret S-box x. The same attack holds for on-the-fly variant as well since the two registers used as part of the computation holds the same set of values as in T_2 over different instances of time.

4 2-O Table Compression Scheme with Pre-processing

In this section, we present our second-order secure lookup table compression scheme. Our approach is to secure the second-order lookup table compression scheme from [Vad17, Section 3] (see Section 3) by first basing it on the modified RDP scheme from Section 2 to allow pre-processing, and then by using different output masks for each of the rows of T_1 and T_2. A part of these output masks are generated using the pair-wise independent PRG technique discussed in Section 2.2.

4.1 Computing T_1 Offline

The first temporary table T_1 is computed as follows. For an index $a^{(1)} \in \{0,1\}^{n-l}$, we have

$$T_1(b^{(1)}) \longleftarrow \text{s-sum}(b^{(1)}) \oplus \bigoplus_{0 \le i \le 2^l-1} \mathrm{S}_{(i \oplus w)}((a^{(1)} \oplus r_{(i \oplus x_1^{(2)})}) \oplus \upsilon^{(1)}), \text{ where} \tag{10}$$

$$b^{(1)} \longleftarrow a^{(1)} \oplus ((x_1^{(1)} \oplus \upsilon^{(1)}) \oplus x_2^{(1)}). \tag{11}$$

Similar to the original scheme [Vad17, Algorithm 6], a random variable $v^{(1)}$ (see (5)) is used to mask x_1 while combining with the second share x_2. Random variables r_i's are assigned values from a set of uniform random distribution that is independent of the secret x. Each S_i lookup function (see (3)) is shifted with a distinct r_i, for $0 \leq i \leq 2^{n-l}$, and each entry of T_1 *packs* 2^l such entries. The same set of r_i's are passed as input to T_2 that will be used to securely extract the required value from the pack of 2^l values.

To conceal the mapping between S_i and r_i, a random variable $w \in \{0,1\}^l$ is used with $x_1^{(2)}$ to shift r_i's (see Remark 2). Moreover, this mapping decides the index of first output share y_1 (see Remark 3). While packing, $v^{(1)}$ is used in combination with shifted r_i's such that the third share is needed only while accessing T_1 during the construction of T_2 in the online phase. To ensure security, the computation has to be carried out in the same order as mentioned (see Remark 4). As in Section 2, the output masks are computed using a pair-wise independent PRG based on the subset-sum technique (Algorithm 2). We stress here that, as we did in Section 2, we are not storing the output masks, but instead masks are computed on-the-fly even when they are needed for the second time. Output masks are used to randomise the rows of T_1 (see Remark 5). For compactness, we refer to the s-sum construction of $b^{(1)}$ as s-sum$(b^{(1)})$. Since any two out of three shares can be picked uniform random and independent of x (see (1)), T_1 can be constructed offline. Algorithm 4 summarises the steps involved in the construction of T_1.

Algorithm 4: Computing T_1 offline.

Input :

- Input shares: x_1, x_2, and $v^{(1)}$ (see (5))
- r_i, $0 \leq i \leq 2^l - 1$
- An (n, m) S-box look up functions S_i, where $\mathrm{S}_i(y) = \mathrm{S}(y \parallel i)$ for $0 \leq i \leq 2^l - 1$
- $w \in \{0,1\}^l$
- $\gamma_j \in \{0,1\}^m$, for $0 \leq j \leq n - l$

Output : Table T_1

```
1  $d_1 \longleftarrow (x_1^{(1)} \oplus v^{(1)}) \oplus x_2^{(1)}$
2  for $a^{(1)} \leftarrow 0$ to $2^{n-l} - 1$ do
3      $b^{(1)} \longleftarrow a^{(1)} \oplus d_1$
4      $z \longleftarrow$ s-sum$(n - l, b^{(1)}, \gamma_0, \ldots, \gamma_{n-l})$
5      $temp \longleftarrow z$
6      for $i \leftarrow 0$ to $2^l - 1$ do
7          $temp \longleftarrow temp \oplus \mathrm{S}_{(w \oplus i)}((a^{(1)} \oplus r_{(x_1^{(2)} \oplus i)}) \oplus v^{(1)})$
8      end
9      $T_1(b^{(1)}) \longleftarrow temp$
10 end
```

Remark 2. The mapping between S_i and r_i decides which r_i to use to lookup T_1 while computing T_2. To ensure that the index of r does not leak any information about x, we shift the r_i's with $x_1^{(2)} \oplus w$. Finally, the same mask is used in T_2 to retain the mapping.

Remark 3. If r_i's are used without shifting the indices, the index of r used in first output share will be $x^{(2)}$, which is a first order leakage of the secret x. To prevent leakage, we can use the mask $(x_1^{(2)} \oplus w)$ to shift index i in r_i's. Then the output share y_1's index is

$((x_3^{(2)} \oplus w) \oplus x_2^{(2)})$. But the pair of intermediate variables $(((x_3^{(2)} \oplus w) \oplus x_2^{(2)}),\ x_1^{(2)} \oplus w)$ together depends on $x^{(2)}$. Therefore, we cannot naïvely combine the values. In the proposed scheme, we shift r_i's indices with $x_1^{(2)}$ and lookup functions S_i index with w in T_1 construction.

Remark 4. Note that the packing of 2^l values of S_i, whose inputs are shifted by r_i, leaks the combined sum of all the r_i's. This combined sum can be used with an intermediate variable from T_2 to leak the secret x. We thwart this leakage by using the output mask $y_{1,i}$ appended while packing the values. Also, the order of evaluation has to be strictly maintained to prevent such attacks.

Remark 5. Note that we do not use the randomness optimisation for y_2's and r_i's. Since the compression scheme suggested is efficient in practice only for small values of $l = 1, 2, 3$ (see Table 5), in such cases we need just a tiny amount of randomness (see Table 3). Also, because the r_i's are combined as part of the construction of T_1 and T_2, pair-wise independence is no longer sufficient, and k-wise independent PRG construction for a large k will bring in a significant overhead compared to using a TRNG.

4.2 Computing T_2 Online

Similar to T_1, the lower bits of shares are combined using $v^{(2)}$ which is derived from a uniform random and independent value v (see (5)). Table T_2 is indexed by $b^{(2)}$ as in (8). Out of 2^l randomised S_i values compressed in T_1, we need to extract the value corresponding to $\mathrm{S}_i(x^{(1)})$, $0 \le i \le 2^l - 1$. To do this, the idea is to combine all the S_i values except the one with index i. Finally, to construct the Table T_2 in such a way that each row is independent of the secret x, we need to randomise the rows of the table. Since the same output masks y_1 and y_2 are used across the Table T_2 in [Vad17, Algorithm 8], the attack mentioned in (9) is possible as two rows of T_2 depend on the secret bits $x^{(1)}$. Hence we use distinct output masks for each row. Algorithm 5 describes the steps of calculating T_2. Third input share x_3 combined with r_i is used to lookup T_1. Index i in r_i's is shifted the same way as in T_1 (see Remark 2):

$$T_1(x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}), \quad \text{where } p = ((x_3^{(2)} \oplus w) \oplus x_1^{(2)}), \quad \forall\, 0 \le a^{(2)} \le 2^l - 1. \tag{12}$$

The value obtained in (12) is combined with all except one S_i that are indexed by shares of $x^{(1)}$ and r_i's to obtain the entries of table T_2. Finally $\mathrm{S}(x) \oplus y_1 \oplus y_2$ can be obtained by a table lookup of T_2 at $v^{(2)}$:

$$y_3 = T_2(v^{(2)}). \tag{13}$$

Algorithm 6 lists the steps of our second-order lookup table compression scheme.
Correctness: the following equations prove that the Algorithm 6 results in the correct sharing of S-box evaluation. Since the Table T_1 uses an output mask that is constructed using the unique index per row, the index of the first output share when $b^{(2)} = v^{(2)}$ is $x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})}$. The final output share returned by Algorithm 6 satisfies:

$$\begin{aligned} y_3 &:= T_2(v^{(2)}) \\ &\quad T_1(x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})}) \oplus y_{2,v^{(2)}} \oplus \\ &\quad \bigoplus_{0 \le j \le 2^l - 1,\ j \ne (x_1^{(2)} \oplus x_2^{(2)})} \mathrm{S}_{(x_3^{(2)} \oplus j)}\big(x^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})} \oplus r_{(((x_3^{(2)} \oplus w) \oplus x_1^{(2)}) \oplus j)}\big) \end{aligned}$$

Algorithm 5: Computing T_2 online.

```
Input :
    • Input shares: $x_1$, $x_2$, $x_3 := x \oplus x_1 \oplus x_2$, and $\upsilon^{(2)}$ (see (5))
    • $r_i$, $0 \le i \le 2^l - 1$
    • An $(n, m)$ S-box look up functions $\mathrm{S}_i$, where $\mathrm{S}_i(y) = \mathrm{S}(y \parallel i)$
      for $0 \le i \le 2^l - 1$
    • $w \in \{0,1\}^l$
    • $y_{2,j} \in \{0,1\}^m$, for $0 \le j \le 2^l - 1$
    • Table $T_1$
Output : Table $T_2$
1  $d_2 \longleftarrow (x_1^{(2)} \oplus \upsilon^{(2)}) \oplus x_2^{(2)}$
2  $p \longleftarrow (x_3^{(2)} \oplus w) \oplus x_1^{(2)}$
3  for $a^{(2)} \leftarrow 0$ to $2^l - 1$ do
4      $b^{(2)} \longleftarrow a^{(2)} \oplus d_2$
5      $temp \longleftarrow T_1(x_3^{(1)} \oplus r_{(p \oplus a^{(2)})})$
6      $sxor \longleftarrow y_{2,b^{(2)}} \oplus temp$
7      for $j \leftarrow 0$ to $2^l - 1$ do
8          if $j \neq a^{(2)}$ then
9              $sxor \leftarrow sxor \oplus$
10                 $\mathrm{S}_{(x_3^{(2)} \oplus j)}\Big(\big(((x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}) \oplus x_1^{(1)}) \oplus r_{(p \oplus j)}\big) \oplus x_2^{(1)}\Big)$
11         end
12     end
13     $T_2(b^{(2)}) \longleftarrow sxor$
14 end
```

$$
\begin{aligned}
&= \text{s-sum}(x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})}) \oplus y_{2,\upsilon^{(2)}} \oplus \\
&\quad \bigoplus_{0 \le i \le 2^l - 1} \mathrm{S}_{(w \oplus i)}\big(x^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})} \oplus r_{(x_1^{(2)} \oplus i)}\big) \oplus \\
&\quad \bigoplus_{0 \le j \le 2^l - 1,\ j \neq (x_1^{(2)} \oplus x_2^{(2)})} \mathrm{S}_{(x_3^{(2)} \oplus j)}\big(x^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})} \oplus r_{(((x_3^{(2)} \oplus w) \oplus x_1^{(2)}) \oplus j)}\big) \\
&= \text{s-sum}(x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})}) \oplus y_{2,\upsilon^{(2)}} \\
&\quad \bigoplus_{0 \le k \le 2^l - 1} \mathrm{S}_k\big(x^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})} \oplus r_{(x_1^{(2)} \oplus w \oplus k)}\big) \\
&\quad \bigoplus_{0 \le k \le 2^l - 1,\ k \neq x^{(2)}} \mathrm{S}_k\big(x^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})} \oplus r_{(x_1^{(2)} \oplus w \oplus k)}\big) \\
&= \mathrm{S}_{x^{(2)}}(x^{(1)}) \oplus \text{s-sum}(x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})}) \oplus y_{2,\upsilon^{(2)}}
\end{aligned}
$$

which proves the correctness of Algorithm 6.

Algorithm 6: Second-order lookup table compression scheme.

```
Input  :
    • Three input shares $x_1,\ x_2,\ x_3 := x \oplus x_1 \oplus x_2$
    • An $(n, m)$ S-box look up functions $\mathrm{S}_i$, where $\mathrm{S}_i(y) = \mathrm{S}(y \parallel i)$
      for $0 \leq i \leq 2^l - 1$
Output : Output shares: $y_1,\ y_2$ and $y_3 := \mathrm{S}(x) \oplus y_1 \oplus y_2$
1  $\upsilon \xleftarrow{\$} \{0,1\}^n$
2  $w \xleftarrow{\$} \{0,1\}^l$
3  for $i \leftarrow 0$ to $n - l$ do
4      $\gamma_i \xleftarrow{\$} \{0,1\}^m$
5  end
6  for $i \leftarrow 0$ to $2^l - 1$ do
7      $r_i \xleftarrow{\$} \{0,1\}^{n-l}$
8      $y_{2,i} \xleftarrow{\$} \{0,1\}^m$
9  end
10 Create table $T_1$ using Algorithm 4.
11 Create table $T_2$ using Algorithm 5.
12 $y_1 := \text{s-sum}(n - l, (x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})}), \gamma_0, \ldots, \gamma_{n-l})$
13 $y_2 := y_{2,\upsilon^{(2)}}$
14 $y_3 := T_2(\upsilon^{(2)})$
15 return $y_1,\ y_2,\ y_3$
```

4.3 2-SNI Security Proof

The intuition behind the proof is that any two output shares are easy to simulate since the two output masks are chosen uniform random and independent. All the intermediate variables that are part of pre-processing can be simulated independent of x, i.e., using only two shares x_1 and x_2. Any combination of the probed pair of rows from tables T_1 and T_2 can be assigned random values as the table rows are randomised by using pair-wise independent output masks. For the pair of variables involving all three shares, we can use the fact that any two input shares are always combined with the help of an independent random value that is not probed. Therefore, the unprobed randomness can act as an one-time pad in simulation of those pairs that depend on the secret x. But as explained below, the simulation of pairs of intermediate variables using the shared randomness (w and r_i's) across T_1 and T_2, is indeed the non-trivial part of the proof. Though the shared randomness between T_1 and T_2 helps reduce the randomness usage and also save computation time, it prevents us from proving the 2-SNI property of the constructions of T_1 and T_2 independently and allow for a trivial composition. However, we are able to directly show that the composition of T_1 and T_2 is 2-SNI.

For completeness, the list of intermediate variables of Algorithm 6 are grouped and tabulated in Table 2. Each I_k represents input or output or intermediate variables as defined in Table 2. We have partitioned the set of intermediate variables of the scheme into four sets :

$$\begin{aligned}\mathrm{Set}_1 &:= \text{constants and random variables},\\ \mathrm{Set}_2 &:= \text{variables of } T_1 \text{ Algorithm},\\ \mathrm{Set}_3 &:= \text{variables of } T_2 \text{ Algorithm},\\ \mathrm{Set}_4 &:= \text{output shares}.\end{aligned}$$

Theorem 2. *Algorithm 6 is 2-SNI secure.*

Proof. Formally, to prove our second-order table compression scheme to be 2-SNI secure, we consider the gadget to be the given S-box S, which takes its input x in the form of three shares $x_1, x_2, x_3 = x \oplus x_1 \oplus x_2$ and outputs y_1, y_2, $y_3 = \mathrm{S}(x) \oplus y_1 \oplus y_2$.

To prove the 2-SNI security (see Definition 2), the following are the possible types of the combinations for probing pairs of intermediate variables (including input shares, intermediate variables and output shares).

1. *Pair of output variables*: if the probed output variables are (y_1, y_2) or (y_2, y_3) or (y_1, y_3), then the pair can be assigned uniform random and independent values.

2. *One input and one output*: similar to Step 1, it is trivial to simulate the probed output share. To simulate the other probed intermediate variable, we must use at most one input share. If the probed intermediate variable depends on at most one input share (including constants, sampled randomness and input shares), then it is straightforward to simulate it using the corresponding input share. It can be observed from Algorithm 6 (and also Table 2) that any variable depending on two or more input shares always consists of an unprobed random variable such as v, w, r's, subset-sums of γ's, or y_2's that acts as one-time pad. Hence we can conclude that any variable from Set_1 or Set_2 or Set_3 can be simulated with a maximum of one input share.

3. *Two intermediate variables*: for this combination, by the definition of 2-SNI, we need to prove that any pair of variables (excluding the output variables) can be assigned values using at most two input shares. All the probed pairs of intermediate variables depending on at most two input shares can be trivially simulated, which indeed implies that any pair from T_1 construction (see Algorithm 4) can be assigned/computed using two input shares. The probed pair depending on all the three shares can only be simulated with the help of unprobed randomness present in the variables. But, if the pair of variables are using shared random variables, then the unprobed randomness can not be treated as one-time pad. Therefore, the simulation of those pairs depending on all the three input shares along with shared randomness, as one may encounter in the construction of T_2, needs to be addressed separately. Following is the case by case analysis of simulation of these pairs, ordered by their occurrence in the construction of T_2 (Algorithm 5).

   - Probing the pair $T_1(x_3^{(1)} \oplus r_{(p\oplus a^{(2)})})$ and the output mask for this row of T_1 (i.e. s-sum$(x_3^{(1)} \oplus r_{(p\oplus a^{(2)})})$) is not possible since the index of T_1 is randomised.
   - Simulation of pair of rows of T_1 indexed using x_3 in Algorithm 5: assign uniform random and independent values to r_i's and w as this would have happened in the actual implementation. Consider the values of $a^{(2)}$ at which the rows are probed, say i, and j. Then, compute $k_1 = x_3^{(1)} \oplus r_{\left(\left((x_3^{(2)}\oplus w)\oplus x_1^{(2)}\right)\oplus i\right)}$ and $k_2 = x_3^{(1)} \oplus r_{\left(\left((x_3^{(2)}\oplus w)\oplus x_1^{(2)}\right)\oplus j\right)}$ using two input shares x_1 and x_3 and return a

Table 2: List of variables of Algorithm 4.

j	I_j
Constants and random values	
1	$c \in \{\text{constant}, \text{loop variable}\}$
2	r, a random value $\in \{\upsilon, w, \gamma_i, y_{(2,j)}$ and r_k's$\}$
Variables of T_1 (see Algorithm 4)	
3	input shares x_1 and x_2
4	$\upsilon^{(1)}$
5	$x_1^{(1)} \oplus \upsilon^{(1)}$
6	$d_1 = (x_1^{(1)} \oplus \upsilon^{(1)}) \oplus x_2^{(1)}$
7	$b^{(1)} = a^{(1)} \oplus d_1$
8	$x_1^{(2)} \oplus j$
9	$r_{(x_1^{(2)} \oplus j)}$
10	$(a^{(1)} \oplus r_{(x_1^{(2)} \oplus j)}) \oplus \upsilon^{(1)}$
11	$\mathrm{S}_{(w \oplus j)}((a^{(1)} \oplus r_{(x_1^{(2)} \oplus j)}) \oplus \upsilon^{(1)})$
12	$\mathrm{S}_{(w \oplus i)}((a^{(1)} \oplus r_{(x_1^{(2)} \oplus i)}) \oplus \upsilon^{(1)}),\ 0 \leq j \leq 2^l - 1$
13	$z \longleftarrow \text{s-sum}(b^{(1)})$
14	$(z) \oplus \bigoplus_{0 \leq i \leq k} \mathrm{S}_{(w \oplus i)}((a^{(1)} \oplus r_{(x_1^{(2)} \oplus i)}) \oplus \upsilon^{(1)}),\ 0 \leq k \leq 2^l - 1$
15	$T_1(b^{(1)}) = (z) \oplus \bigoplus_{0 \leq j \leq 2^l - 1} \mathrm{S}_{(w \oplus i)}((a^{(1)} \oplus r_{(x_1^{(2)} \oplus i)}) \oplus \upsilon^{(1)})$
Variables of T_2 (see Algorithm 5)	
16	input shares x_1, x_2 and $x_3 = x \oplus x_1 \oplus x_2$
17	$\upsilon^{(2)}$
18	$x_1^{(2)} \oplus \upsilon^{(2)}$
19	$d_2 = (x_1^{(2)} \oplus \upsilon^{(2)}) \oplus x_2^{(2)}$
20	$b^{(2)} = a^{(2)} \oplus d_2$
21	$x_3^{(2)} \oplus w$
22	$p = (x_3^{(2)} \oplus w) \oplus x_1^{(2)}$
23	$q = (x_3^{(2)} \oplus w) \oplus x_2^{(2)}$
24	$r_{(p \oplus a^{(2)})}$
25	$x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}$
26	$(x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}) \oplus x_1^{(1)}$
27	$((x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}) \oplus x_1^{(1)}) \oplus r_{(p \oplus j)}$
28	$x^{(1)} \oplus r_{(p \oplus a^{(2)})} \oplus r_{(p \oplus j)}$
29	$\mathrm{S}_{(x_3^{(2)} \oplus j)}(x^{(1)} \oplus r_{(p \oplus a^{(2)})} \oplus r_{(p \oplus j)}),\ j \neq a^{(2)}$
30	$T_1(x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}) = \text{s-sum}(x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}) \oplus \bigoplus_{0 \leq i \leq 2^l - 1} \mathrm{S}_{(x_3^{(2)} \oplus a^{(2)})}(x^{(1)} \oplus r_{(p \oplus a^{(2)})} \oplus r_i)$

31	$(y_{2,b^{(2)}}) \oplus \bigoplus_{0 \le j \le k,\ j \ne a^{(2)}} \mathrm{S}_{(x_3^{(2)} \oplus j)}(x^{(1)} \oplus r_{(p \oplus a^{(2)})} \oplus r_{(p \oplus j)}),\ 0 \le k \le 2^l - 1$
32	$T_2(b^{(2)}) = \mathrm{S}_{(x_3^{(2)} \oplus a^{(2)})}(x^{(1)}) \oplus \text{s-sum}(x_3^{(1)} \oplus r_{(p \oplus a^{(2)})}) \oplus y_{2,b^{(2)}}$
Output shares	
33	$y_1 := \text{s-sum}(x_3^{(1)} \oplus r_{((x_3^{(2)} \oplus w) \oplus x_2^{(2)})})$
34	$y_2 = y_{2,v^{(2)}}$
35	$y_3 = T_2(v^{(2)}) = \mathrm{S}\ (x) \oplus y_1 \oplus y_2$

uniform random and independent value for $T(k_1)$ and $T(k_2)$ if $k_1 = k_2$, return two uniform random and independent values, otherwise.

- Along with intermediate variable $x^{(1)} \oplus r_{(p \oplus a^{(2)})} \oplus r_{(p \oplus j)}$, $j \ne a^{(2)}$ either only $r_{(p \oplus a^{(2)})}$ or $r_{(p \oplus j)}$ can be probed, since r_i's are not combined directly as part of Algorithm 5. Therefore, the unprobed r can be used for the simulation of this pair. No knowledge of input shares is needed for the simulation of this pair.
- Probing the partial sum of S-box lookup functions S_i at two distinct indices, say $i = k_1$ and $i = k_2$ (see I_{31}) will have the same output mask y_2. The simulation of this pair proceeds as follows: assign uniform random and independent values to y_2 and r_i's. Since r_i is unique per $\mathrm{S}_{(x_3^{(2)} \oplus a^{(2)})}(x^{(1)} \oplus r_i \oplus r_j)$, $r_i \ne r_j$ lookup function, r_i can be used to randomise $x^{(1)} \oplus r_i \oplus r_j$, an $(n-l)$-bit value. Compute $x_3^{(2)} \oplus a^{(2)}$, the lower order l-bits, with the help of the input share x_3 and loop variable $a^{(2)}$. Then evaluate the corresponding S-box values by concatenating the $n - l$ and l-bit strings. Finally, compute the probed pair using y_2 and S-box values. Note that this argument does not require S-box to be balanced (see Definition 1). This simulation requires only one input share, x_3.
- While computing the indices of r_i's in T_2, the shares are combined using the mask w. And the same w is used while computing the index of final output share. But, the order of combination is taken care (see I_{22} and I_{23}), therefore using the random mask w and two input shares x_1 and x_2, the indices of r_i's can be computed.
- Any pair of rows from T_2 can be assigned uniform random and independent values without using input shares since the rows are randomised using distinct output masks.

This proves our claim that the table compression scheme presented in Algorithm 6 is 2-SNI secure. □

4.4 Memory Complexity

For the scheme proposed in Algorithm 6, the memory requirement for masking a single S-box lookup table is as follows: Table T_1 has 2^{n-l} entries each of size m bits, Table T_2 has 2^l entries each of size m bits. The 2^l random r_i's are $(n-l)$ bits each. The set of $n - l$ random values used to generate first output masks are m bits each, and so is the set of 2^l second output masks $y_{2,i}$. Therefore the total memory needed, parametrised by n and l, for the compression scheme proposed in Algorithm 6 is $m \cdot (n-l+1) + 2^l \cdot (n-l+m) + m \cdot (2^{n-l} + 2^l)$ bits. Table 3 details the memory requirement for any (8,8)-bit S-box, in particular, for AES-128. For completeness, Table 4 details the memory requirement for any (4,4)-bit S-box, in particular, for PRESENT.

Table 3: Estimate of RAM memory required for Algorithm 6 for an (8,8)-bit S-box. Memory requirement (in no. of bytes) for r_i's, y_1's, y_2's, T_1, T_2, are also listed separately. The last column corresponds to the total number of random bytes required for r_i, y_1 and y_2's.

l	r_i	y_1	y_2	T_1	T_2	Total Memory	Randomness
1	2	8	2	128	2	142	12
2	3	7	4	64	4	82	14
3	5	6	8	32	8	59	19
4	8	5	16	16	16	61	29
5	12	4	32	8	32	88	48
6	16	3	64	4	64	151	83
7	16	2	128	2	128	276	146

Table 4: Estimate of RAM memory required for Algorithm 6 for a (4,4)-bit S-box. Memory requirement (approximated in no. of bytes) for r_i's, y_1's, y_2's, T_1, T_2, are also listed separately. The last column corresponds to the total number of random bytes required for r_i, y_1 and y_2's.

l	r_i	y_1	y_2	T_1	T_2	Total Memory	Randomness
1	1	2	1	4	1	9	4
2	1	2	2	2	2	9	5
3	1	1	4	1	4	11	6

5 Implementation Results

We have performed a second-order masked software implementation of the full block ciphers AES-128 [FIP] and PRESENT 80-bit key variant [BKL$^+$07]. Our source code is available at [VV]. The randomised S-box table compression is achieved using the compression scheme we proposed in Section 4. Our target architecture is NXP-FRDM-k64F, an ultra-low-cost development platform. The FRDM-K64 is supported by various open source embedded operating systems. The microcontroller used in the development platform is MK64FN1M0VLL12, a low-power microcontroller based on ARM Cortex-M4 processor having a 256 KB RAM, 1 MB flash memory and a clock frequency of 120 MHz. For AES, we have built our code on top of the publicly available masked implementations from [Cor]. For PRESENT, we referred the publicly available unmasked implementation from [Klo]. The code size for full cipher implementation of AES-128 and PRESENT using our proposed scheme is 12.8 KB and 8.5 KB, respectively.

We have computed the ratio of resulting execution times with the respective unmasked implementations of AES-128 and PRESENT. The computed ratio is expressed as *penalty factor*, PF. Since the compression parameter l has an impact on the time vs. memory requirements of these algorithms, we ran these experiments for l ranging from 1 to $n-1$. In order to achieve security for the full block cipher implementation, the randomness is sampled using the RNGA module in-built to the micro-controller, which generates 32-bit random number in approximately 300 clock cycles.

For an efficient implementation, it is better to pre-process T_1 for every S-box function call (see Remark 6). The number of pre-computed T_1 tables required are same as the number of S-box invocations. Hence, the number of tables required are 160 (16/round $\cdot$ 10 rounds) and 496 (16/round $\cdot$ 31 rounds) for AES-128 and PRESENT block ciphers, respectively.

The tables are pre-processed and stored on the same chip where the actual execution takes place. It is easy to see that pre-processing helps to reduce the online computation by about a factor of 2^{n-l} for arbitrary (n, m) S-boxes. For instance, the amount of RAM memory required to store the pre-processed tables is 8.1 KB for the optimal case $l = 3$ (see Table 5), while outperforming [RP10] in the online execution time. Needless to say, for highly resource constrained devices, pre-processing may not be feasible and in this case all the computations will happen during the online phase to keep the peak memory usage to a minimum.

There is a trade-off between online execution time and RAM (volatile memory) that results in the choice between constructing T_1 online vs. pre-processing T_1. This gives us two possibilities for time vs. memory trade-offs.

1. Compute T_1 online and T_2 using table-based computation for each S-box invocation.
2. Pre-process and store T_1 for all the S-box invocations, compute T_2 using table-based computation for each S-box invocation.

Tables 5 and 6 presents the implementation results for the Option 2 above for various levels of compression for AES and PRESENT, respectively. Since the number of bytes required to store the tables T_1 and T_2 vary with the compression parameter, the total memory required for the full cipher execution (that includes the memory required for pre-processing, online execution) is listed for each value of l. The execution times are given in seconds. The penalty factor is for the online execution time. We would like to mention that the time for pre-processing refers to the computation which happens before the start of block cipher execution (*not* at the time of compilation), which is independent of the actual input. By online time, we mean the time required for a *single* block cipher execution where the actual inputs are needed.

Table 5: Second-order masked S-box compression using the table-based scheme with pre-processing (see Algorithm 6). Total memory required for AES-128 in KB and execution times are in seconds.

l	Total Memory	Offline Time	Online Time	Total Time	Penalty Factor
1	22.5	0.039241	0.001507	0.040748	11.8363
2	12.7	0.025615	0.002321	0.027936	18.2297
3	8.1	0.019431	0.005719	0.02515	44.9183
4	5.4	0.017248	0.010648	0.027896	83.6318
5	4.6	0.017495	0.010651	0.028146	83.6554
6	4.3	0.020253	0.010638	0.030891	83.5533
7	3.9	0.026813	0.010652	0.037465	83.6632

Table 6: Second-order masked S-box compression using the table-based scheme with pre-processing (see Algorithm 6). Total memory required for PRESENT in KB and execution times are in seconds.

l	Total Memory	Offline Time	Online Time	Total Time	Penalty Factor
1	4.1	0.009156	0.005955	0.015111	8.3873
2	3.2	0.007545	0.008642	0.016187	12.1718
3	2.1	0.007687	0.010713	0.0184	15.0887

For the second-order circuit based scheme, there is no offline computation advantage which implies the total-time is nothing but the online time. The RAM memory usage is very little for these schemes. Table 7 indicates the timings for second-order [RP10] scheme with *FullRefresh* [DDF14b] that is proven t-SNI secure in [BBD$^+$16]. The table also indicates timings for the bitsliced masked implementation of AES [RSD06, GR17] adapted to 8-bit architecture with three shares.

Table 7: Second-order S-box implementation of AES-128 with three input shares using the second-order circuit based schemes [RP10] with FullRefresh and 8-bit bitslicing. The execution times are in seconds. The RAM memory required is in bytes.

Scheme	Online Time	Total Time	Memory	Penalty Factor
[RP10]	0.010638	0.010638	24	83.5533
8-bit bitsliced masking	0.008108	0.008108	996	63.682

Table 8 presents timings for the second-order masked lookup table scheme [RDP08], its variant with pre-processing (see Algorithm 1), and the higher-order lookup table scheme from [CRZ18, Algorithm 3] instantiated at second order. Recall that the original scheme from [RDP08] does not allow pre-processing and hence its online time is the sum of online and offline time for Algorithm 1 but the RAM memory needed is very small. It may be observed that even though the penalty factor is only around 5 for Algorithm 1, the size of total RAM memory required is about 40 KB.

Table 8: Second-order instantiation of masked lookup table implementation of AES using [CRZ18], the RDP scheme and its variant with pre-processing (see Algorithm 1). Total RAM memory required for AES-128 is in KB and the execution times are in seconds.

Scheme	Total Memory	Offline Time	Online Time	Total Time	Penalty Factor
[RDP08]	0.3	0	0.00721	0.00721	56.629
Algorithm 1	40.6	0.006604	0.000606	0.00721	4.76
Second-order [CRZ18]	120.4	0.089411	0.017792	0.107203	139.742

In Table 9 we present the implementation results for second-order masked PRESENT using [CRV15]. We would like to note that for the implementations of second-order PRESENT using our compression scheme and the circuit-based CRV scheme, we processed only 4-bit values, one at a time. However, to minimise the memory required for S-box table compression, we choose to pack two 4-bit values into a byte. Therefore, we assume that probing any intermediate variable during masked S-box computation will only leak 4-bit values. Since the lookup table compression and circuit-based implementations are based on 4-bit variants, we think that it may not be fair to compare the results with 8-bit bitsliced masked implementation of PRESENT.

Table 9: Second-order S-box implementation of PRESENT using the second-order circuit based scheme [CRV15] with three input shares. The execution times are in seconds. The memory required is in bytes.

Online Time	Total Time	Memory	Penalty Factor
0.008959	0.008959	40	12.6183

For the Option 1 above, the execution times are the total time reported in Table 5 and 6. This is essentially the sum of pre-processing and online execution times. Specifically, for 128-AES, the memory requirement would closely correspond to the figures reported in Table 3. Note that for this case the online execution time is same as the total time.

It can be concluded from the above results of AES-128 (see Table 5) that pre-processing helps to reduce the online execution time, for $1 \leq l \leq 3$. The penalty factors for online execution times when the compression parameter $l \geq 4$ are close to the circuit-based schemes. Hence $l \leq 3$ has practical relevance. In our implementations we have not explored packing multiple bytes into the 32-bit register. We chose not to do this optimisation because for architectures with small register size (for e.g., 8-bit microcontrollers) such a packing is not possible.

Remark 6. To achieve better online timings, T_1 can be pre-processed and stored. But, if the same T_1 is used across all S-box function calls, probing x_3 across S-box executions is going to leak the *xor* of the secrets. Therefore, we compute a separate table T_1 per S-box invocation to secure the full cipher implementation. This also helps to prevent horizontal side-channel attacks.

6 Conclusion

Since first-order DPA attacks on embedded devices are feasible, in this work we focused on designing a second-order secure masked table compression scheme for highly resource constrained embedded devices to achieve time-memory trade-offs. In spite of the RAM memory reduction, our online execution time is still better than circuit-based masking schemes. Though higher-order masking is very costly for highly resource-constrained devices, yet it will be an interesting research direction to design masked table compression schemes for any order. We saw that our second-order table compression scheme is more involved than the first-order schemes from [Vad17, Viv17]. This complexity is mainly because of the compression of table rows and then secure expansion at a particular index, and this can be challenging to handle at arbitrary orders. It will also be interesting to explore alternate techniques to achieve time-memory trade-offs at higher orders.

Acknowledgements

This work was funded by the INSPIRE Faculty Award (DST, Govt. of India) for Srinivas Vivek. We thank K K Subramaniam and Girish S Kumar for their inputs regarding development platform. We also thank Shyam Murthy for helpful review comments.

References

[BBD+15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified Proofs of Higher-Order Masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

[BBD+16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong Non-

Interference and Type-Directed Higher-Order Masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 116–129. ACM, 2016.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.

[BKL$^+$07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[CGP$^+$12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-Order Masking Schemes for S-Boxes. In Anne Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 366–384. Springer, 2012.

[CGZ19] Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel Masking with Pseudo-Random Generator. *IACR Cryptology ePrint Archive*, 2019:1106, 2019. To appear at EUROCRYPT 2020.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In Wiener [Wie99], pages 398–412.

[Cor] Jean-Sébastian Coron. Higher-order countermeasures for AES and DES. Available at `https://github.com/coron/htable`. Last accessed on April 15, 2020.

[Cor14] Jean-Sébastien Coron. Higher Order Masking of Look-Up Tables. In Nguyen and Oswald [NO14], pages 441–458.

[Cor18] Jean-Sébastien Coron. Formal Verification of Side-channel Countermeasures via Elementary Circuit Transformations. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2018.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic Decomposition for Probing Security. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015, Proc., Part I*, volume 9215 of *LNCS*, pages 742–763. Springer, 2015.

[CRV15] Jean-Sébastien Coron, Arnab Roy, and Srinivas Vivek. Fast Evaluation of Polynomials over Binary Finite Fields and Application to Side-channel Countermeasures. *J. Cryptographic Engineering*, 5(2):73–83, 2015.

[CRZ18] Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High Order Masking of Look-up Tables with Common Shares. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):40–72, 2018.

[DDF14a] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In Nguyen and Oswald [NO14], pages 423–440.

[DDF14b] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440. Springer, 2014.

[FH17] Wieland Fischer and Naofumi Homma, editors. *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*. Springer, 2017.

[FIP] NIST FIPS. Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, US Department of Commerce/NIST, November 26, 2001. Available from the NIST website.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How Fast Can Higher-Order Masking Be in Software? In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

[GRVV17] Dahmun Goudarzi, Matthieu Rivain, Damien Vergnaud, and Srinivas Vivek. Generalized Polynomial Decomposition for S-boxes with Application to Side-Channel Countermeasures. In Fischer and Homma [FH17], pages 154–171.

[GTP$^+$20] Zhipeng Guo, Ming Tang, Emmanuel Prouff, Maixing Luo, and Fei Yan. Table Recomputation-Based Higher-Order Masking Against Horizontal Attacks. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 39(1):34–44, 2020.

[IKL$^+$13] Yuval Ishai, Eyal Kushilevitz, Xin Li, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and David Zuckerman. Robust Pseudorandom Generators. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, volume 7965 of *Lecture Notes in Computer Science*, pages 576–588. Springer, 2013.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.

[JS17] Anthony Journault and François-Xavier Standaert. Very High Order Masking: Efficient Implementation and Security Evaluation. In Fischer and Homma [FH17], pages 623–643.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Wiener [Wie99], pages 388–397.

[Klo] D. Klose. C PRESENT Implementation. Available at `http://www.lightweightcrypto.org/implementations.php`. Last accessed on July 5, 2020.

[Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *CRYPTO 1996, Proc.*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[NO14] Phong Q. Nguyen and Elisabeth Oswald, editors. *EUROCRYPT 2014. Proc.*, volume 8441 of *LNCS*. Springer, 2014.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against Side-Channel Attacks: A Formal Security Proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013. Proc.*, volume 7881 of *LNCS*, pages 142–159. Springer, 2013.

[RDP08] Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In Kaisa Nyberg, editor, *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, volume 5086 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2008.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010. Proc.*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[RRST02] Josyula R. Rao, Pankaj Rohatgi, Helmut Scherzer, and Stephane Tinguely. Partitioning Attacks: Or How to Rapidly Clone Some GSM Cards. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 31–41. IEEE Computer Society, 2002.

[RSD06] Chester Rebeiro, A. David Selvakumar, and A. S. L. Devi. Bitslice implementation of AES. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *Cryptology and Network Security, 5th International Conference, CANS 2006, Suzhou, China, December 8-10, 2006, Proceedings*, volume 4301 of *Lecture Notes in Computer Science*, pages 203–212. Springer, 2006.

[SP06] Kai Schramm and Christof Paar. Higher Order Masking of the AES. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, 2006.

[TV09] Tamir Tassa and Jorge L. Villar. On proper secrets, (t , k)-bases and linear codes. *Des. Codes Cryptogr.*, 52(2):129–154, 2009.

[Vad17] Praveen Kumar Vadnala. Time-Memory Trade-Offs for Side-Channel Resistant Implementations of Block Ciphers. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 115–130. Springer, 2017.

[Viv17] Srinivas Vivek. Revisiting a Masked Lookup-Table Compression Scheme. In Arpita Patra and Nigel P. Smart, editors, *Progress in Cryptology - INDOCRYPT 2017 - 18th International Conference on Cryptology in India, Chennai, India, December 10-13, 2017, Proceedings*, volume 10698 of *Lecture Notes in Computer Science*, pages 369–383. Springer, 2017.

[VV] Annapurna Valiveti and Srinivas Vivek. Implementation of Second-order Table Compression. Available at `https://github.com/annapurna-pvs/second-order-table-compression`. Last accessed on July 7, 2020.

[Wie99] Michael J. Wiener, editor. *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*. Springer, 1999.